```
=> dis his

     (FILE 'HOME' ENTERED AT 17:00:20 ON 23 FEB 2001)

     FILE 'USPATFULL' ENTERED AT 17:00:28 ON 23 FEB 2001
L1             0 S US6601764/PN
L2             0 S US6601764/PN

     FILE 'USPATFULL' ENTERED AT 17:01:45 ON 23 FEB 2001
L3             1 S US6061764/PN
L4         19249 S CACHE#
L5       1192926 S SIZE#
L6             1 S L4 AND L3
L7             1 S L5 AND L3
L8       1816458 S LINE###
L9             1 S L3 AND L8
L10       389239 S VARIABLE
L11            1 S L3 AND L10
L12       106089 S STATUS
L13            1 S L3 AND L12
L14        34100 S HIT
L15            1 S L4 AND L3

=> s l14 and l3

L16            0 L14 AND L3

=> s tag

L17        25835 TAG

=> s tag#

L18        31636 TAG#

=> s l18 and l3

L19            0 L18 AND L3

=> s field#

L20     1483467 FIELD#

=> s l20 and l3

L21            1 L20 AND L3

=> dis hit


L21  ANSWER 1 OF 1  USPATFULL
PI     US 6061764  20000509                                          <--
SUMM   In a serial bus 70, messages are exchanged among agents to implement a
       transaction. A first message carries a request from one agent to
       another. A second message carries a response to the request back to the
       requesting agent. Several transactions may be underway at once on the
       serial bus. Messages include a transaction ID field to
```

identify the transaction to which the message relates.

SUMM On the serial bus, read request messages identify the request type. The message also includes an address of data to which the request is directed and a length of data to be read. The address identifies a starting point for reading data; the length **field** indicates how much data should be read from memory beginning at that address.

SUMM Read requests typically are answered by a response that provides the requested data. The response message typically includes a header that includes a response type, a transaction ID and a length **field** identifying the length of the data that follows. When the read request cannot be fulfilled for any reason, the request may be answered by a RETRY response.

DETD The inbound path includes an inbound transaction queue 130 and a splitter 140. The inbound transaction queue 130 stores transaction information that has been decoded by the format decoder 112. Typically, the format decoder 112 decodes a message into a format appropriate for the pipelined bus 60. The inbound transaction queue 130 also stores data that is inappropriate for the pipelined bus 60, such as the transaction ID and the length **field** received from a read request message. This information is used internally.

DETD The splitter 140 reads transactions from the inbound transaction queue 130 and, where appropriate, presents them to the bus request generator 122 to be posted on the pipelined bus 60. For queued read request, the splitter 140 examines the length **field** and issues atomic transactions or non-atomic transaction.

DETD The read request is received from the serial bus 70 by the format decoder 112, decoded into a format suitable for the pipelined bus and queued in the inbound transaction queue 130 as described above. When the read request advances out of the inbound transaction queue 130, the splitter 140 examines the address and length **field** and determines whether the requested data spans more than one cache line. If the requested data falls within a single cache line, the splitter 140 issues a single non-atomic transaction. If the requested data spans more than one cache line, the splitter 140 issues a multiple number of atomic transactions.

DETD The bus request generator 122 also passes the transaction ID, the length **field** and the true lower order address bits to the bus request decoder 124 internally.

DETD The bus request decoder 124 observes the transaction on the pipelined bus 60 and receives the transaction ID, length **field** and true address bits from the bus request generator 122. In response to the transaction, it stores the transaction information in the outbound transaction queue 150. The bus request decoder also allocates a cache line of space in the staging buffer 160 and stores a pointer to the cache line in the outbound transaction queue 150.

DETD Data is read from the staging buffer 160 one cache segment at a time. To access an appropriate cache segment, the controller 164 is initialized with the pointer identifying the cache line, lower order bits of the true address and the length **field**. The address bits identify the first cache segment to be read. The controller 164 causes the selection switch 166 to pass the selected cache segment to the outbound FIFO queue 170. Based on the length **field**, the controller 164 advances to the adjacent cache segments and reads them out of the staging buffer 160.

DETD The outbound transaction controller 180 generates a response and outputs

it to the format encoder 114. It also causes the queued data stored in the outbound FIFO queue 170 to be read to the format encoder 114. Where data is read out in data segment increments, the outbound transaction controller 170 identifies a first data segment based on the true address bits. The outbound transaction controller 180 causes the selection switch 174 to pass a selected data segment to the format encoder 114. The outbound transaction controller 180 advances to a next adjacent data segment and reads it out. The outbound transaction controller 180 continues to read out data segments until it traverses the length of data specified in the length **field**.

DETD As noted, the splitter 140 observes the transaction as it advances out of the inbound transaction queue 130. From the length **field**, the splitter 140 determines that the transaction requests data of multiple cache lines. In response, the splitter 140 issues atomic transactions to the bus request generator 122. With one exception, the splitter enables an atomic flag with each atomic transaction. The atomic flag is disabled for the final atomic transaction in the series. Thus, the beginning and end of a series of atomic transactions is indicated by the status of the atomic flag.

DETD The bus request decoder 124 observes the atomic transactions on the pipelined bus 60. It also receives from the bus request generator 122 the atomic flags of the atomic transactions along with the transaction ID, the length **field** and the true lower order address bits. The bus request decoder 124 decodes atomic transactions as it would any read transaction issued by the MIOC 100 on the pipelined bus 60. It allocates data in the staging buffer 160 and stores the transaction information in the outbound transaction queue 150.

DETD At the conclusion of the final atomic transaction, administrative data of each atomic transaction is stored in the outbound transaction queue 150. The outbound transaction queue 150 stores the transaction ID, the length **field** and the true value of the lower order address bits with the administrative data of the first atomic transaction. The staging buffer 160 stores data associated with each atomic transaction.

DETD The MIOC 100 reads data out of the outbound FIFO queue 170 to the format encoder 114. The transaction ID and the length **field** is read to the format encoder 114 first. Thereafter, the cache segments are read out of the outbound FIFO queue 170. As with the non-atomic transaction, reading of data may begin at a certain data segment within the first queued cache segment. The lower order address bits are used by the outbound transaction controller 180 to select a first data segment to be read out. The outbound transaction controller 180 causes the data segment to be passed through the selection switch 176 then advances to the next data segment. The outbound transaction controller 180 continues to read data segments and advance until it has traversed the number of data segments identified by the length **field**. When the last data segment of a cache segment is reached, advancing to the next data segment causes a wrap to the beginning of a next cache segment.

DETD The format encoder 114 receives the transaction ID, length **field** and retrieved data from the outbound FIFO queue 170. It formats the information into a response message and outputs it to the serial bus 70.

=> s address?

L22 256191 ADDRESS?

=> s l22 and l3

L23 1 L22 AND L3

L13 ANSWER 1 OF 69 USPATFULL
PI US 6301647 B1 20011009
TI Real mode translation look-aside buffer and method of operation

L13 ANSWER 2 OF 69 USPATFULL
PI US 6298411 B1 20011002
TI Method and apparatus to share instruction images in a virtual cache

L13 ANSWER 3 OF 69 USPATFULL
PI US 6298367 B1 20011002
TI Floating point addition pipeline including extreme value, comparison and
accumulate functions

L13 ANSWER 4 OF 69 USPATFULL
PI US 6266752 B1 20010724
TI Reverse TLB for providing branch target address in a microprocessor
having a physically-tagged cache

L13 ANSWER 5 OF 69 USPATFULL
PI US 6256653 B1 20010703
TI Multi-function bipartite look-up table

L13 ANSWER 6 OF 69 USPATFULL
PI US 6240484 B1 20010529
TI Linearly addressable microprocessor cache

L13 ANSWER 7 OF 69 USPATFULL
PI US 6223256 B1 20010424
TI Computer cache memory with classes and dynamic selection of replacement
algorithms

L13 ANSWER 8 OF 69 USPATFULL
PI US 6223192 B1 20010424
TI Bipartite look-up table with output values having minimized absolute
error

L13 ANSWER 9 OF 69 USPATFULL
PI US 6212629 B1 20010403
TI Method and apparatus for executing string instructions

L13 ANSWER 10 OF 69 USPATFULL
PI US 6175898 B1 20010116
TI Method for prefetching data using a micro-TLB

L13 ANSWER 11 OF 69 USPATFULL
PI US 6161208 20001212
TI Storage subsystem including an error correcting cache and means for
performing memory to memory transfers

L13 ANSWER 12 OF 69 USPATFULL
PI US 6157993 20001205
TI Prefetching data using profile of **cache misses** from
earlier code executions

L13 ANSWER 13 OF 69 USPATFULL
PI US 6138226 20001024
TI Logical cache memory storing logical and physical address information

for resolving synonym problems

L13 ANSWER 14 OF 69 USPATFULL
PI US 6131104 20001010
TI Floating point addition pipeline configured to perform floating point-to-integer and integer-to-floating point conversion operations

L13 ANSWER 15 OF 69 USPATFULL
PI US 6094708 20000725
TI Secondary cache write-through blocking mechanism

L13 ANSWER 16 OF 69 USPATFULL
PI US 6094668 20000725
TI Floating point arithmetic unit including an efficient close data path

L13 ANSWER 17 OF 69 USPATFULL
PI US 6088715 20000711
TI Close path selection unit for performing effective subtraction within a floating point arithmetic unit

L13 ANSWER 18 OF 69 USPATFULL
PI US 6085212 20000704
TI Efficient method for performing close path subtraction in a floating point arithmetic unit

L13 ANSWER 19 OF 69 USPATFULL
PI US 6085208 20000704
TI Leading one prediction unit for normalizing close path subtraction results within a floating point arithmetic unit

L13 ANSWER 20 OF 69 USPATFULL
PI US 6079005 20000620
TI Microprocessor including virtual address branch prediction and current page register to provide page portion of virtual and physical fetch address

L13 ANSWER 21 OF 69 USPATFULL
PI US 6079003 20000620
TI Reverse TLB for providing branch target address in a microprocessor having a physically-tagged cache

L13 ANSWER 22 OF 69 USPATFULL
PI US 6065091 20000516
TI Translation look-aside buffer slice circuit and method of operation

L13 ANSWER 23 OF 69 USPATFULL
PI US 6047363 20000404
TI Prefetching data using profile of **cache misses** from earlier code executions

L13 ANSWER 24 OF 69 USPATFULL
PI US 6041396 20000321
TI Segment descriptor cache addressed by part of the physical address of the desired descriptor

L13 ANSWER 25 OF 69 USPATFULL
PI US 6038644 20000314
TI Multiprocessor system with partial broadcast capability of a cache coherent processing request

L13 ANSWER 26 OF 69 USPATFULL
PI US 6032241 20000229
TI Fast RAM for use in an address translation circuit and method of operation

L13 ANSWER 27 OF 69 USPATFULL
PI US 6014728 20000111
TI Organization of an integrated cache unit for flexible usage in supporting multiprocessor operations

L13 ANSWER 28 OF 69 USPATFULL
PI US 5991902 19991123
TI Memory apparatus and data processor using the same

L13 ANSWER 29 OF 69 USPATFULL
PI US 5970509 19991019
TI Hit determination circuit for selecting a data set based on miss determinations in other data sets and method of operation

L13 ANSWER 30 OF 69 USPATFULL
PI US 5963984 19991005
TI Address translation unit employing programmable page size

L13 ANSWER 31 OF 69 USPATFULL
PI US 5960463 19990928
TI Cache controller with table walk logic tightly coupled to second level access logic

L13 ANSWER 32 OF 69 USPATFULL
PI US 5954435 19990921
TI Memory apparatus and data processor using the same

L13 ANSWER 33 OF 69 USPATFULL
PI US 5953520 19990914
TI Address translation buffer for data processing system emulation mode

L13 ANSWER 34 OF 69 USPATFULL
PI US 5930833 19990727
TI Logical cache memory storing logical and physical address information for resolving synonym problems

L13 ANSWER 35 OF 69 USPATFULL
PI US 5918062 19990629
TI Microprocessor including an efficient implemention of an accumulate instruction

L13 ANSWER 36 OF 69 USPATFULL
PI US 5913224 19990615
TI Programmable cache including a non-lockable data way and a lockable data
way configured to lock real-time data

L13 ANSWER 37 OF 69 USPATFULL
PI US 5867724 19990202
TI Integrated routing and shifting circuit and method of operation

L13 ANSWER 38 OF 69 USPATFULL
PI US 5781753 19980714
TI Semi-autonomous RISC pipelines for overlapped execution of RISC-like instructions within the multiple superscalar execution units of a processor having distributed pipeline control for speculative and out-of-order execution of complex instructions

L13 ANSWER 39 OF 69 USPATFULL
PI US 5778427 19980707
TI Method and apparatus for selecting a way of a multi-way associative cache by storing waylets in a translation structure

L13 ANSWER 40 OF 69 USPATFULL
PI US 5768575 19980616